Simple Guide to Using GNU AutoTools

by Travis Parker and R. Paul Ritchey

ARL-CR-0681 **October 2011**

prepared by:

ICF Jacob and Sundstrom
401 E. Pratt Street, Suite 2214
Baltimore, MD 21202-3003

under contract

W911QX-07-F-0023

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005

ARL-CR-0681 **October 2011**

Simple Guide to Using GNU AutoTools

Travis Parker and R. Paul Ritchey
Computational and Information Sciences Directorate, ARL

prepared by:

ICF Jacob and Sundstrom
401 E. Pratt Street, Suite 2214
Baltimore, MD 21202-3003

under contract

W911QX-07-F-0023

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
October 2011	Final	October 2009 to August 2010

4. TITLE AND SUBTITLE
Simple Guide to Using GNU AutoTools

5a. CONTRACT NUMBER
W911QX-07-F-0023

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Travis Parker and R. Paul Ritchey*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ICF Jacob and Sundstrom
401 E. Pratt Street, Suite 2214
Baltimore, MD 21202-3003

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-CR-0681

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CIN-D
Aberdeen Proving Ground, MD 21005

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
*ICF Jacob and Sundstrom

14. ABSTRACT
While working on porting a C based project to use the GNU AutoTools for configuration and compilation we discovered that the guides available online were either outdated or not clear enough for someone unfamiliar with AutoTools. After successfully adding AutoTools to that project and many others, we thought other programmers may benefit from what we learned and the basic starter files that we created that make starting a new project from scratch or adding AutoTools to an existing project a snap. This report is not intended to be an in depth guide to all of the possible features and capabilities AutoTools provides. It is intended to provide the information and basic AutoTool configuration files needed for a typical project.

15. SUBJECT TERMS
GNU AutoTools, software development, compiling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	24	Chuck Smith
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER (*Include area code*) (410) 278-6235

Contents

1. What are GNU AutoTools? 1

2. Directory Structure and AutoTool Files 1

3. Sample Project: Simple_Project 7

4. Sample Project: Simple_Project2 7

5. Sample Project: Simple_Project3 8

6. Conclusion 9

Appendix A. Source Code Listings – Simple Project 11

Appendix B. Source Code Listings – Simple Project 2 15

Appendix C. Source Code Listings – Simple Project 3 17

Distribution List 18

INTENTIONALLY LEFT BLANK.

1. What are GNU AutoTools?

GNU AutoTools is a set of tools (AutoMake, AutoConf, libtool, etc.) used to automate the build process for software projects. Most Linux distributions come with some or all of the tools already installed, and if you've downloaded and built an open source project from source code you've used the tools without realizing it when you installed it ('./configure ; make ; make install'). In addition to automating the build process, the tools allow you to define configuration items that can be dynamically enabled or disabled during compilation and make your project more easily transported between distributions by allowing it to automatically locate the libraries required by your code. [1]

For most distributions you can easily determine if the AutoTools are installed by either using the GUI package manager provided by your distribution and searching for 'auto' or using a command line utility. For example, from the command line of an RPM based distribution, you can easily determine if the AutoTools is installed by issuing the following command and reviewing the output:

```
rpm -qa | grep auto
automake-x.x.x-x.x
autoconf-x.x-x
...
```

In the above sample output, both *automake* and *autoconf* are installed. If they weren't, you would use the mechanism provided by your distribution (GUI package manager or command line) to install them and their required dependencies. Follow the same steps to see if libtool is installed if your project builds shared libraries.[2]

2. Directory Structure and AutoTool Files

The directory structure we chose for our projects consists of the main project directory and two subdirectories. The two subdirectories contain the source code files (*.c, *.cpp) in one directory and the include files (*.h) in the other. For some of our projects, where the source code is broken up into multiple directories (such as those including internal libraries), keeping the include files in a separate, shared directory has allowed us to use the same base AutoTool configuration files for numerous projects with very few changes beyond the normal changes needed to customize them for a specific project.

[1] GNU Build System: http://en.wikipedia.org/wiki/GNU_build_system

[2] Manpage of rpm(8): http://linux.die.net/man/8/rpm

```
simple_project
|-- include
`-- src
```

Basic directory structure of a project.

Once the directory structure is created, you need to populate it with a specific set of files. The listing below shows each directory with the basic files that are needed (including source code) by AutoTools. The files will be described later, including a review of their contents and what needs to be changed to customize them for each new project.

```
simple_project
|-- AUTHORS
|-- ChangeLog
|-- Makefile.am
|-- NEWS
|-- README
|-- autogen.sh
|-- configure.ac
|-- include
|   `-- simple_project.h
`-- src
    |-- Makefile.am
    `-- simple_project.c
```

The directories populated with the basic files.

The `Authors`, `ChangeLog`, `NEWS` and `REAME` files are simple text files. Initially you can create empty files using the touch command ('`touch AUTHORS ChangeLog NEWS README`') and then keep them updated as appropriate as work on your project progresses. If you project is for internal use and you have no plans on using them you will still need to create them – otherwise the `automake` command will display an error message about their absence.

> *Note: Complete listings of the files for the AutoTools configuration files and scripts discussed in this document can be found in the Appendices. Snippets of these files are provided throughout the document for easy reference while reading the text.*

The `autogen.sh` file contains a very simple but important shell script that executes three of the autotool commands to dynamically build the `configure` script and input files it needs using the configuration files you create. **It's very important to remember to re-run this script after you make a change to any of the AutoTool configuration files**. If you forget to run it after making a change, the `configure` script and its output will not be updated with your latest changes. It's also been our experience that this script should be run again when moving between different distributions because they may not have the same version of AutoTools installed. After the script is executed, you will see several new files created throughout your directories. These files can be ignored by you as they are automatically generated and do not contain anything that should be altered; they will be overwritten the next time you run the `autogen.sh` script. If a

change to one of the files is necessary, you need to make the change in the appropriate file listed above and re-run the `autogen.sh` script. This script should be included in the top level of your source code directory and can be reused without any changes.

The `Makefile.am` file, located in your project's root directory, is one of the simpler configuration files you'll need to update for each of your projects. The entry `SUBDIRS` is the only configuration entry that you will need to change and should contain a space delimited list of all of the subdirectories containing source code that requires compilation. If there is more than one directory, they should appear in the same order in which they need to be compiled because of dependencies. In the example provided below, there is only one source code directory.[3]

```
# Macro file for use with GNU AutoTools for generating config files and Makefiles.
# This file was manually created based on information found via Google and the
# GNU documentation.  Although it works for the source code it's included with,
# it may not be 100% correct so additional research/tweaking may be needed.
#

# AUTOMAKE_OPTIONS is used to pass options to 'automake'.  In this case
# we're telling it to use the GNU tool chain.
AUTOMAKE_OPTIONS = gnu

# SUBDIRS takes a space delimited list of _all_ the subdirectories that
# contain source code.
SUBDIRS = src
```

File Listing: Makefile.am

The `configure.ac` file, also located in your project's base directory, is significantly longer than `Makefile.am` and has several lines that require customization for your projects. The first entry requiring a change is `AC_INIT` which contains two entries. The first entry is the name of your project which will be used to name the final binary when the compilation process has been completed. The second entry is the version number. In the example below, a binary named `simple_project` will be built and the version is 1.0.

```
# AC_INIT takes two parameters, the package name and a version number.
AC_INIT([simple_project], [1.0])
```

The AC_INIT parameter is used to specify the binary name and version.

The next parameter in the file that needs to be altered is the `AC_CONFIG_SRCDIR`. It requires one parameter, the name of a single source code file in your project and the path where it's located. When updating it for our projects, we usually specify the file containing the `main()` function in our C/C++ code.

```
# AC_CONFIG_SRCDIR takes one parameter, the path and filename
# for one of the source code files.
AC_CONFIG_SRCDIR([src/simple_project.c])
```

The AC_CONFIG_SRCDIR needs to point to a source code file and the path to it.

[3] GNU Automake: http://www.gnu.org/software/hello/manual/automake/index.html

Skipping to the bottom of the `configure.ac` file, you'll see the `AC_OUTPUT` parameter. Depending on your project's requirements, this may be the last item you need to update. In order for AutoTools to know where to build `Makefiles`, you must specify them using a space delimited list in the `AC_OUTPUT` parameter. The root directory of your project must be included in the list and this is accomplished by simply including '`Makefile`' in the list. The rest of the entries should include the word `Makefile` pre-pended with the directory. In the example below, the root directory of our project is included along with one subdirectory, `src`, which is the only directory containing source code that needs to be compiled.[4]

```
# AC_OUTPUT takes a space delimited list of where the Makefiles are to be created.
# You need to pass all directories where there is source code (including the base
# directory containing all of the source code (which won't need a path pre-pended
# to the 'Makefile' keyword).
AC_OUTPUT(Makefile src/Makefile)
```

You must supply a list of directories where all of the source code can be found.

Some projects require additional libraries, or allow you to enable/disable features using the configure script. By adding one additional line per library/optional feature to the `configure.ac` script, the proper checks and tests will be performed up front instead of during compilation. It will also display a more user friendly message for the user, making it easier for them to resolve the issue. This is implemented using a macro named `REQUIRE_LIB` which is located approximately halfway into the `configure.ac` file.

The macro takes four parameters. The first parameter is the name of the library as it exists in the file system without any extensions. The sample line below is for the libpcap library. The actual name of the file is `libpcap.so`, so the first parameter should simply be '`libpcap`'. The second parameter will be appended to the text '`--with-<second_parameter>`' which is displayed in the configure script's help output and is used as an option to enable/disable a feature or provide the path to the library if the configure script is unable to find it automatically. The third parameter is the name of a function included in the library. During the configuration phase, AutoTools will create a test source code file which includes a call to the supplied function and AutoTools will attempt to compile it to ensure the library is installed and is accessible. The fourth and final parameter is the text you want displayed if the library is missing, inaccessible or the user views the help documentation the configure script provides ('`./configure --help`').

```
REQUIRE_LIB(libpcap,pcap,pcap_dump_open,[Libpcap packet capture library])
```

A sample entry for a library that is required by a project.

```
--with-libpcap=<path>   Location where Libpcap packet capture library is
                        Installed
```

This is how the text included will be used in the configure script's help output.

[4] GNU Autoconf : http://www.gnu.org/software/autoconf/manual/index.html

It is also possible to use the REQUIRE_LIB macro to enable or disable features in your code. To do this, wrap the call to REQUIRE_LIB in an 'if' statement testing for the parameter the REQUIRE_LIB will look for. In this example, the user can enable support for accessing Oracle databases by passing a '--with-liboci' to the configure script. Simply replace the 'liboci' with the name of the library (first parameter passed to REQUIRE_LIB) you want to allow as an option. An AutoTools configure script will set pre-compilation defines ('#define') based on enabled features, which can be used to IFDEF sections of code.

Continuing with the example, if '--with-liboci' is provided during configuration, 'LIBSQLPLUS' will be defined from the combination of 'LIB' plus the uppercase version of the second parameter passed to REQUIRE_LIB.

```
if test "$with_liboci"; then
  REQUIRE_LIB(liboci,sqlplus,OCIEnvCreate,[Oracle OCI client libraries provided with
Instant Client])
fi
```

Wrapping REQUIRE_LIB in an 'if' statement allows you to use it to dynamically enable or disable features.

Now that the files in the base directory of your project have been covered, the next file that needs to be created are the Makefile.in which needs to exist in each of the source code subdirectories listed in the SUBDIRS parameter in the Makefile.am file. In the simple example we've been using so far, the only subdirectory requiring a Makefile.am file is the src directory.

The Makefile.am contains two entries that you will need to update for each project, while the rest are optional. The first parameter, bin_PROGRAMS, defines the name of the binary that should be built at the completion of the compilation process. In the example below, the binary will be named 'simple_project'. The name of the binary will also be pre-pended to other parameters in the file as you will soon see.

```
# bin_PROGRAMS is used to define the binary that's to be built from the
# source code.  The binary name will also be used in variable names to
# pass values that will only apply to this binary.
bin_PROGRAMS = simple_project
```

bin_PROGRAMS should be set to the name of the binary that will be built.

The second parameter that you will need to change is the <binary_name>_SOURCES parameter. You must set the beginning portion of the name of this parameter to the name you used for the binary in the bin_PROGRAMS parameter. In our example the parameter's name becomes simple_project_SOURCES. The space delimited list following this parameter is a complete list of the source code files that need to be compiled. Order can be important depending on the dependencies between the code files, so if you change the order don't forget to re-run the autogen.sh script.

```
# <binary_name>_SOURCES takes a space delimited list of the source code files
# that need to be compiled/linked to build the binary.
simple_project_SOURCES = simple_project.c
```

For this parameter you need to update the name with the name of the binary being built.

The rest of the parameters listed in the example below will need the name of the binary pre-pended as well. They are optional and allow you to provide additional parameters at link time, such as search paths and additional libraries that need to be linked in. This capability is particularly useful when your project contains its own libraries.

```
# <binary_name>_LDADD is used to pass extra parameters at link time, such as
# libraries that need to be linked in.
simple_project3_LDADD = -lfoo

# <binary_name>_LDFLAGS is used to pass extra parameters at compilation time,
# such as the paths to libraries that are needed that are not in the default
# paths.
# If the user supplies a directory where libpcap is installed, the contents
# of the variable setup in configire.in will be substituted below so the
# library file will be found.
simple_project3_LDFLAGS = -L../libfoo
```

If they are used, remember to update the names of these parameters.

The final parameter in the `Makefile.am` file is the `INCLUDE` parameter. This parameter may not need to be updated for different projects. For our projects we centralize all of the header files ('.h') in the '`include`' subdirectory of the project, so for us, this parameter never needs to be updated between projects or between multiple source code directories within the same project.

```
# The INCLUDES macro allows us to tell the tools where needed header files are
# located if they aren't in the default paths.  In this case it's a subdirectory
# in the source code directory where all the header files are centrally located.
# If the user supplies a directory where a library is installed, the contents
# of the variable setup in configure.in will be substituted below so the
# header file will be found.
INCLUDES = -I../include
```

The parameter allows you to specify where in your project the include files can be found.

Once you have reached this point, you should have all of the necessary configuration files AutoTools needs to generate the final versions. Execute the `autogen.sh` script to update the configuration files generated by AutoTools that are based upon the ones you created. If the `autogen.sh` script executed without any errors you will be able to '`./configure ; make ; make install`' to configure, compile and install your project. If autogen.sh does not execute without errors, you will need to make appropriate adjustments to your configuration files and then re-execute `autogen.sh`.

3. Sample Project: Simple_Project

Included in the compressed archive is a project titled 'simple_project'. This example is a very simple project and has been used as the basis for the code examples in this paper. It consists of one include file and one source code file. This project is designed to be the template for a project made up of one or more code files and requiring no libraries (internal or provided by the operating system). Complete listings for the files described previously appear at the end of this document for reference.

4. Sample Project: Simple_Project2

Simple_project2 is based on the first sample project, with the addition of an internal library that gets statically compiled in. The internal library is contained in the '`libfoo`' subdirectory of the source code, requiring slight changes to the `Makefile.am` and `configure.am` in the project's base directory and the `Makefile.am` file in the '`src`' subdirectory. The primary changes of interest are to the '`<binary_name>_LDADD`' and '`<binary_name>_LDFLAGS`' parameters in the `src` directory which contain the flags/paths for where the library can be found.

Changes are also required in the `Makefile.am` file located in the `libfoo` subdirectory. The `lib_LIBRARIES` parameter instructs the AutoTools that the type of library to be built is *not* a shared library, and the parameter value is the name of the library.

```
# The lib_LIBRARIES macro tells the autotools the name and type of library we
# want to build - in this case _not_ a shared library.  A shared library
# would end with '.la' instead of '.a'.
lib_LIBRARIES = libfoo.a
```

Including this parameter informs AutoTools it is to build a non-shared library.

The second change in the `libfoo` subdirectory's `Makefile.am` file is the '`<library_name>_<library_type>_SOURCES`' parameter. The name of this parameter is used to specify both the name and type of library being build, while the value of the parameter is a space delimited list of the source code that must be compiled to create the library. In the sample project the library name is '`libfoo`' which is a static library type ('`.a`') and consists of one source code file ('`libfoo.c`').

```
# This specially named variable contains a space delimited list of the source code
# files that must be compiled in order to create the library.  The variable name
# is in the format:
# <library_name>_<library_type>_SOURCES
# Where:
# <library_name> = the name of the library specified in lib_LIBRARIES
```

```
# <library_type> = either 'a' for non-shared library or 'la' for shared.
libfoo_a_SOURCES = libfoo.c
```

You must also add a parameter that includes the name/type of library and the list of code files.

---

5. Sample Project: Simple_Project3

Simple_project3 is similar to simple_project2, but instead of linking the project's included library in statically, it is linked in as a shared library. To accomplish this, several changes to the AutoTool configuration files in the project needed to be made.

The first unique change is to the `configure.ac` file located in the base directory of the project. Near the top of the file you will see the addition of the `AC_PROG_LIBTOOL` parameter. Including this parameter allows you to build shared libraries.

```
# AC_CONFIG_SRCDIR takes one parameter, the path and filename
# for one of the source code files.
AC_CONFIG_SRCDIR([src/simple_project3.c])
AM_INIT_AUTOMAKE
AC_PROG_LIBTOOL
```

For shared libraries you must add the AC_PROG_LIBTOOL option to configure.ac.

The next changes that are unique to building a shared library is to the `Makefile.am` file in the library's subdirectory. You must use the '`lib_LTLIBRARIES`' parameter to build a shared library whose value is set to the name of the library.

```
# The lib_LIBRARIES macro tells the autotools the name and type of library we
# want to build - in this case a shared library.  A shared library
# ends with '.la' instead of '.a' which is for a non-shared (static) library.
lib_LTLIBRARIES = libfoo.la
```

To build a shared library, use the lib_LTLIBRARIES option in the Makefile.am file.

The next change to the `Makefile.am` is the parameter specifying the space delimited list of source code files that make up the shared library. The name of the parameter follows the format '`<library_name>_<library_type>_SOURCES`', which in this case `<library_type>` is specified as '`la`' for shared library.

```
# This specially named variable contains a space delimited list of the source code
# files that must be compiled in order to create the library.  The variable name
# is in the format:
# <library_name>_<library_type>_SOURCES
# Where:
# <library_name> = the name of the library specified in lib_LIBRARIES
# <library_type> = either 'a' for non-shared library or 'la' for shared.
libfoo_la_SOURCES = libfoo.c
```

Be sure to correctly name the _SOURCES parameter to build a shared library.

6. Conclusion

The GNU Autotools allow developers to use an automate the build process that is supported on a variety of platforms. However, the GNU documentation for Autotools is complex as it must cover all features and cases. In this paper we have presented a set of simplified instructions for converting most projects to use GNU Autotools.

INTENTIONALLY LEFT BLANK.

Appendix A. Source Code Listings – Simple Project

This appendix contains a listing of the major files in the Simple Project's directory. It is not an exhaustive listing of all files that will appear, especially after the AutoTools utilities are executed.

autogen.sh (Located in the project's base directory.)

The autogen.sh script is generic and can be included in any project without any changes. Remember to executed it after any changes are made to the project's AutoTool configuration files.

```
#! /bin/sh

# This script is used to automate the process of running some of the autotools
# against their input files (configure.in, Makefile.am) after _any_ of them have
# been updated.  The commands and parameters were taken based on a similar script
# found via google and seeing the same commands issued in GNU autotool tutorials.

aclocal
automake --add-missing --force-missing
autoconf
```

Makefile.am (Located in the project's base directory.)

```
# Macro file for use with GNU AutoTools for generating config files and Makefiles.
# This file was manually created based on information found via Google and the
# GNU documentation.  Although it works for the source code it's included with,
# it may not be 100% correct so additional research/tweaking may be needed.
#

# AUTOMAKE_OPTIONS is used to pass options to 'automake'.  In this case
# we're telling it to use the GNU tool chain.
AUTOMAKE_OPTIONS = gnu

# SUBDIRS takes a space delimited list of _all_ the subdirectories that
# contain source code.
SUBDIRS = src
```

configure.ac (Located in the project's base directory.)

```
# Macro file for use with GNU AutoTools for generating config files and Makefiles.
# This file was manually created based on information found via Google and the
# GNU documentation.  Although it works for the source code it's included with,
# it may not be 100% correct so additional research/tweaking may be needed.
#

# AC_INIT takes two parameters, the package name and a version number.
AC_INIT([simple_project], [1.0])

# AC_CONFIG_SRCDIR takes one parameter, the path and filename
# for one of the source code files.
AC_CONFIG_SRCDIR([src/simple_project.c])
AM_INIT_AUTOMAKE

AC_PROG_CC
AC_PROG_RANLIB

CFLAGS=
LDFLAGS=
LIBS=

# This is a reusable macro for providing --with-libfoo functionality.
#
```

```
# REQUIRE_LIB(name,lib,testfn,description)
#    name = The complete name of the library file without the extension.
#    lib = The name of the library file without the 'lib' prefix and without the extension.
#    testfn = One function included in the library that can be used for a test compilation.
#    description = Human readable text to be displayed if the library can't be found or
#                  if there's a problem during the test compilation.
AC_DEFUN([REQUIRE_LIB], [ {
        AC_ARG_WITH([$1], AC_HELP_STRING([--with-$1=<path>],[Location where $4 is
installed]),[],[with_$1=default])

        AS_IF( [test "x$with_$1" != xdefault],
        [
              LDFLAGS="$LDFLAGS -L${with_$1}/lib"
              CFLAGS="$CFLAGS -I${with_$1}/include"
        ])

        AC_CHECK_LIB($2,$3,[],
        [
                AC_MSG_ERROR([$4 was not found, try specifying --with-$1])
        ])
} ] )

# The list of libraries required by the source code that are external to
# our code.
#REQUIRE_LIB(libpcap,pcap,pcap_dump_open,[Libpcap packet capture library])


# AC_OUTPUT takes a space delimited list of where the Makefiles are to be created.
# You need to pass all directories where there is source code (including the base
# directory containing all of the source code (which won't need a path pre-pended
# to the 'Makefile' keyword).
AC_OUTPUT(Makefile src/Makefile)
```

src/Makefile.am

```
# Macro file for use with GNU AutoTools for generating config files and Makefiles.
# This file was manually created based on information found via Google and the
# GNU documentation.  Although it works for the source code it's included with,
# it may not be 100% correct so additional research/tweaking may be needed.
#

# bin_PROGRAMS is used to define the binary that's to be built from the
# source code.  The binary name will also be used in variable names to
# pass values that will only apply to this binary.
bin_PROGRAMS = simple_project

# Note: that the following names all start with the binary name defined in
# bin_PROGRAMS.

# <binary_name>_LDADD is used to pass extra parameters at link time, such as
# libraries that need to be linked in.
simple_project_LDADD =

# <binary_name>_LDFLAGS is used to pass extra parameters at compilation time,
# such as the paths to libraries that are needed that are not in the default
# paths.
# If the user supplies a directory where libpcap is installed, the contents
# of the variable setup in configire.in will be substituted below so the
# library file will be found.
simple_project_LDFLAGS =

# <binary_name>_SOURCES takes a space delimited list of the source code files
# that need to be compiled/linked to build the binary.
simple_project_SOURCES = simple_project.c

# The INCLUDES macro allows us to tell the tools where needed header files are
# located if they aren't in the default paths.  In this case it's a subdirectory
# in the source code directory where all the header files are centrally located.
# If the user supplies a directory where libpcap is installed, the contents
# of the variable setup in configire.in will be substituted below so the
# header file will be found.
```

```
INCLUDES = -I../include
```

include/simple_project.h

```
char global_foo[] = "Hello World!";
```

src/simple_project.c

```
#include <stdio.h>
#include <stdlib.h>
#include "simple_project.h"

int main() {
    printf("%s\n", global_foo);
    return 0;

}
```

INTENTIONALLY LEFT BLANK.

Appendix B. Source Code Listings – Simple Project 2

This appendix contains a listing of the major files in the Simple Project 2's directory. It is not an exhaustive listing of all files that will appear, especially after the AutoTools utilities are executed. Several files (such as the base `Makefile.am`) are not included as the modifications that are needed beyond original Simple Project are described earlier in the document.

include/simple_project2.h

```
#ifndef simple_project2_h_
#define simple_project2_h_

char global_foo[] = "Hello World!";

#endif
```

include/libfoo.h

```
#ifndef libfoo_h_
#define libfoo_h_

void display_stuff(char stuff[]);

#endif
```

libfoo/libfoo.c

```
#include <stdio.h>
#include <stdlib.h>

void display_stuff(char stuff[]) {
    printf("%s\n", stuff);
}
```

src/simple_project2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "simple_project2.h"
#include "libfoo.h"

int main() {
    printf("%s\n", global_foo);
    display_stuff("I'm printed via a library call.");
    return 0;
}
```

INTENTIONALLY LEFT BLANK.

Appendix C. Source Code Listings – Simple Project 3

This appendix contains a listing of the major files in the Simple Project 3's directory. It is not an exhaustive listing of all files that will appear, especially after the AutoTools utilities are executed. Several files (such as the base `Makefile.am`) are not included as the modifications that are needed beyond original Simple Project are described earlier in the document.

autogen.sh (Located in the project's base directory.)

Although the autogen.sh script is generic and can be included in any project without any changes, an additional line needs to be added if the project builds a shared library. The listing below includes the additional line. Remember to executed it after any changes are made to the project's AutoTool configuration files.

```
#! /bin/sh

# This script is used to automate the process of running some of the autotools
# against their input files (configure.in, Makefile.am) after _any_ of them have
# been updated.  The commands and parameters were taken based on a similar script
# found via google and seeing the same commands issued in GNU autotool tutorials.

libtoolize --force
aclocal
automake --add-missing --force-missing
autoconf
```

include/simple_project3.h

```
        #ifndef simple_project2_h_
        #define simple_project2_h_

        char global_foo[] = "Hello World!";

        #endif
```

include/libfoo.h

```
        #ifndef libfoo_h_
        #define libfoo_h_

        void display_stuff(char stuff[]);

        #endif
```

libfoo/libfoo.c

```
        #include <stdio.h>
        #include <stdlib.h>

        void display_stuff(char stuff[]) {
            printf("%s\n", stuff);
        }
```

src/simple_project3.c

```
        #include <stdio.h>
        #include <stdlib.h>
        #include "simple_project3.h"
        #include "libfoo.h"

        int main() {
            printf("%s\n", global_foo);
            display_stuff("I'm printed via a library call.");
            return 0;
}
```

NO. OF COPIES	ORGANIZATION
1	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	US ARMY RSRCH LAB ATTN RDRL CIN D T PARKER BLDG 310E RM C53 ABERDEEN PROVING GROUND MD 21005
1	US ARMY RSRCH LAB ATTN RDRL CIN S P RITCHEY BLDG 310E RM C72 ABERDEEN PROVING GROUND MD 21005
4	US ARMY RSRCH LAB ATTN RDRL CIN S C SMITH BLDG 310E RM C39 ABERDEEN PROVINIG GROUND MD 21005
3	US ARMY RSRCH LAB ATTN IMNE ALC HRR MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB ATTN RDRL CIO MT TECHL PUB ADELPHI MD 20783-1197